Approximation Algorithms for Campaign Management

Edith Elkind
Division of Mathematical Sciences
Nanyang Technological University
Singapore

Piotr Faliszewski
Department of Computer Science
AGH Univ. of Science and Technology
Kraków, Poland

November 29, 2010

Abstract

We study electoral campaign management scenarios in which an external party can buy votes, i.e., pay the voters to promote its preferred candidate in their preference rankings. The external party's goal is to make its preferred candidate a winner while paying as little as possible. We describe a 2-approximation algorithm for this problem for a large class of electoral systems known as scoring rules. Our result holds even for weighted voters, and has applications for campaign management in commercial settings. We also give approximation algorithms for our problem for two Condorcet-consistent rules, namely, the Copeland rule and maximin.

1 Introduction

Elections and voting play an important role in the functioning of the modern society. In the standard model of voting, each voter's preferences are represented by a total order over the alternatives (candidates), and some voting rule is used to determine the election winner(s). However, in practice, the voters' preferences are often flexible, and it is possible to affect the outcome of the election by campaigning for or against a certain candidate. Indeed, campaign management is a multi-million dollar industry, and there is overwhelming evidence that the amount of money invested into a candidate's campaign is strongly correlated with her chances of winning the election.

The notion of *bribery* proposed by Faliszewski, Hemaspaandra, and Hemaspaandra [6] can be viewed as a formal model of electoral campaign management. In the model of [6], each (possibly weighted) voter is associated with a certain price, and, by paying the price, the briber can change that voter's vote in any way she likes. The briber's goal, then, is to get a particular candidate elected, subject to a budget constraint. To connect this description with our original campaign management scenario, observe that bribing a (weighted) voter can be interpreted as mounting an election campaign targeted at a particular group of voters with identical preferences.

However, this interpretation does not take into account that in practice it may be relatively easy to convince a voter to make small changes to his vote, but hard or impossible

1

to convince him to adopt an entirely new preference ordering. To remedy this, several subsequent papers [5, 7, 4] allow the briber to modify the voters' preferences in a more fine-grained manner. In [5] and [7], the authors depart from the assumption that the voters are represented by the preference orders. We will therefore focus on the framework of [4], which operates in the standard model of voting, and assumes that the briber can pay each voter to swap any two candidates that are adjacent in that voter's ordering; this type of bribery is called *swap bribery*. In the context of campaign management, such a swap corresponds to an ad that compares two particular candidates. A special case of swap bribery that was also suggested in [4] is *shift bribery*, where the briber is limited to buying swaps that involve her preferred candidate; in effect, this is equivalent to allowing the briber to shift her preferred candidate up in the voters' preference orderings. The constraint that a campaign ad should involve the briber's preferred candidate is very natural from the ethics perspective; as we will see later, it also leads to more tractable computational problems.

The complexity-theoretic study of swap and shift bribery was initiated in [4], where the authors show that the associated computational problem is hard for many voting rules. However, campaign management can be naturally viewed as an optimization problem, and hence we can approach it using the framework of approximation algorithms. This line of research was first suggested in [4], where the authors give a 2-approximation algorithm for shift bribery under the Borda rule. We expand the study of approximation algorithms for campaign management to voting rules other than Borda, and to weighted voters.

Our main result is a 2-approximation algorithm for shift bribery under all scoring rules (a large class of voting rules, which includes Borda); our result holds even for weighted voters. Under a scoring rule, each candidate gets a certain number of points from each voter, which is determined by that candidate's position in the voter's preferences, and the winner is the candidate with the maximum number of points. Unlike most of the existing algorithms for scoring rules (see, e.g., [6]), our algorithm does not assume that the number of candidates is constant, but rather accepts the scoring vector as an input. Our proof has an unusual structure: we first design a pseudopolynomial 2-approximation algorithm for our problem, then convert it into a $(2 + \varepsilon)$-approximation scheme, and finally turn the $(2 + \varepsilon)$-approximation scheme into a 2-approximation algorithm using a bootstrapping argument.

Interestingly, shift bribery under scoring rules provides mathematical framework for campaign management scenarios that are not related to elections. Consider, for example, an advertiser in a sponsored search setting who wants to ensure that his ads get more clicks than those of the competitors, and is willing to make an additional investment in his campaign to achieve that. By associating the competing ads with candidates, search terms with (weighted) voters, and scores for position i with clickthrough rates for an ad in position i, we can reduce the advertiser's problem to shift bribery with weighted voters. This example suggests that our 2-approximation algorithm can be used for campaign management in a variety of settings, including—but not limited to—voting.

We also give approximation algorithms for shift bribery under two voting rules that have

the attractive property of *Condorcet consistency*, namely, the Copeland rule and maximin.[1]
To complement our positive results, we show that, in contrast to shift bribery, swap bribery
is usually hard to approximate. We conclude the paper by suggesting directions for future
work.

2 Preliminaries

In this section we describe relevant notions from computational social choice and define the
shift bribery problem. We take $\mathbb{Z}^+$ to be the set of all nonnegative integers.

Elections. An *election* is a pair $E = (C, V)$, where $C = \{c_1, \ldots c_m\}$ is the set of *candidates*
and $V = (v^1, \ldots, v^n)$ is a collection of *voters*. Each voter v^i is described by her *preference
order* $\succ^i$, which is a strict linear order over C: $c \succ^i c'$ means that voter v^i prefers c to c'.
We will also consider settings where each voter v^i has a *weight* w_i; in this case, her vote is
interpreted as w_i votes.

A *voting rule* is a function that given an election $E = (C, V)$ outputs a set $W \subseteq C$
of *election winners*. Note that we do not require the voting rule to produce a unique
winner, i.e., we work in the so-called *nonunique-winner model*. This approach is standard
in computational social choice literature, as it allows to define classic voting rules (see
below) in a more natural manner. In practice, this means that a voting rule may have to
be combined with a tie-breaking rule.

Voting Rules. We will now describe several well-known voting rules that will be consid-
ered in this paper. All voting rules listed below are defined for an election $E = (C, V)$ with
$C = \{c_1, \ldots, c_m\}$, $V = (v^1, \ldots, v^n)$. For all rules defined in terms of scores (points), the
winner(s) are the candidate(s) with the maximum score (highest number of points).

Scoring rules. A *scoring rule* $\mathcal{R}_{\vec{\alpha}}$ is described by a vector $\vec{\alpha} = (\alpha_1, \ldots, \alpha_m)$, where $\alpha_i \in \mathbb{Z}^+$
for $i = 1, \ldots, n$, and $\alpha_1 \geq \cdots \geq \alpha_m$. Under $\mathcal{R}_{\vec{\alpha}}$, each candidate c_i receives α_j points
from each voter that ranks him in the j-th position. Note that each scoring rule is
defined for a fixed number of candidates. Thus, we often consider voting rules that
are defined by *families* of scoring rules $(\vec{\alpha}^m)_{m=1,2,\ldots}$, with one vector for each number
of candidates. In particular, *Borda* is the rule given by $\alpha_j^m = m - j$ for $j = 1, \ldots, m$,
and k-*approval* is the rule given by $\alpha_j^m = 1$ for $j \leq k$, $\alpha_j^m = 0$ for $j > k$; 1-approval is
also known as *plurality*.

Condorcet consistent rules. For any $c_i, c_j \in C$, let $N_E(c_i, c_j)$ denote the number of
voters in E who prefer c_i to c_j. If $N_E(c_i, c_j) > N_E(c_j, c_i)$, then we say that c_i wins
the *pairwise election* against c_j. A candidate $c \in C$ is called the *Condorcet winner*
if he wins the pairwise elections against all other candidates in C. Note that some
elections may not have a Condorcet winner. We say that a voting rule $\mathcal{R}$ is *Condorcet-
consistent* if for any election E that has a Condorcet winner c we have $\mathcal{R}(E) = \{c\}$.

---

[1]We mention that an earlier version of this paper also included results on Bucklin voting. Unfortunately,
the proofs contained errors and thus we removed the section on Bucklin from the paper.

Two examples of Condorcet-consistent rules are Copeland and maximin, defined as follows. For any rational $\alpha \in [0,1]$, $Copeland^\alpha$ grants one point to a candidate $c_i \in C$ for each pairwise elections that c_i wins, and α points for each pairwise election that c_i ties. The *maximin score* of c_i is the number of votes that c_i receives in her worst pairwise election, i.e., $\min_{c_j \in C \setminus \{c_i\}} N_E(c_i, c_j)$.

We denote by $Sc_E^{\mathcal{R}}(c)$ the $\mathcal{R}$-score of a candidate $c \in C$ in an election $E = (C, V)$; we omit the superscript $\mathcal{R}$ when the voting rule is clear from the context.

The Shift Bribery Problem. This section is based on the definitions from [4]. Consider an election $E = (C, V)$ with $C = \{p, c_1, \ldots, c_{m-1}\}$, $V = (v^1, \ldots, v^n)$. Suppose that our goal is to ensure that the designated candidate p is a winner of the election under a voting rule $\mathcal{R}$. In order to achieve this goal, we can ask each voter v^i to shift p upwards in her vote by a certain number of positions. This models the fact that we can campaign in favor of p. However, each such shift has a cost. Specifically, each voter v^i has a *cost function* $\pi^i : \mathbb{Z}^+ \to \mathbb{Z}^+$, where $\pi^i(k)$, $k \in \mathbb{Z}^+$, is the cost of shifting p upwards by k positions in $\succ^i$. We require that each π^i, $i = 1, \ldots, n$, satisfies $\pi^i(0) = 0$ and $\pi^i(k) \leq \pi^i(k+1)$ for $k \in \mathbb{Z}^+$. Also, when v^i ranks p in position t, we require $\pi^i(s) = \pi^i(t-1)$ for all $s \geq t$; thus, the function π^i is fully specified by its values at $1, \ldots, t-1$. Note that we assume that $\pi^i(k) < \infty$ for all $i = 1, \ldots, n$ and $k \in \mathbb{Z}^+$. However, all our proofs can be generalized to the case where π^i can be $+\infty$ (i.e., some voters cannot be bribed to move p by more than some given number of positions). We seek an action that makes p a winner at the minimum cost.

Definition 2.1. Let $\mathcal{R}$ be a voting rule. An instance of $\mathcal{R}$-SHIFT-BRIBERY problem is a tuple $I = (C, V, \Pi, p)$, where $C = \{p, c_1, \ldots, c_{m-1}\}$, $V = (v^1, \ldots, v^n)$ is a collection of preference orders over C, $\Pi = (\pi^1, \ldots, \pi^n)$ is a family of cost functions, and $p \in C$ is a designated candidate. The goal is to find a minimal value b such that there is a sequence $\mathbf{t} = (t_1, \ldots, t_n) \in (\mathbb{Z}^+)^n$ with the following properties: (a) $b = \sum_{i=1}^n \pi^i(t_i)$, and (b) if for each $i = 1, \ldots, n$ we shift p upwards in the i-th vote by t_i positions, then p becomes an $\mathcal{R}$-winner of E. We denote this value of b by $\mathrm{opt}(I)$.

In WEIGHTED $\mathcal{R}$-SHIFT-BRIBERY, the description of the instance includes a vector of voters' weights $\mathbf{w} = (w_1, \ldots, w_n)$, i.e., we have $I = (C, V, \Pi, p, \mathbf{w})$.

We will call the sequence $\mathbf{t} = (t_1, \ldots, t_n)$ a *shift-action*. Let $shf(C, V, \mathbf{t})$ denote the election obtained from (C, V) by shifting p upwards by t_i positions in the i-th vote (or placing p on top of that vote, if v^i ranks p in position $t < t_i + 1$ before the bribery). A shift-action is *successful* if p is a winner of $shf(C, V, \mathbf{t})$. Additionally, let $\Pi(\mathbf{t}) = \sum_{i=1}^n \pi^i(t_i)$.

Let $I = (C, V, \Pi, p)$ be an instance of $\mathcal{R}$-SHIFT-BRIBERY and let $\mathbf{t} = (t_1, \ldots, t_n)$. Overloading notation, we let $shf(I, \mathbf{t})$ denote an instance $\hat{I} = (C, \hat{V}, \hat{\Pi}, p)$ of $\mathcal{R}$-SHIFT-BRIBERY given by (a) $(C, \hat{V}) = shf(C, V, \mathbf{t})$, and (b) $\hat{\Pi} = (\hat{\pi}^1, \ldots, \hat{\pi}^n)$ where for each $i = 1, \ldots, n$ we have $\hat{\pi}^i(k) = \pi^i(k + t_i) - \pi^i(t_i)$. That is, $shf(I, \mathbf{t})$ represents the instance of shift bribery obtained from I by applying the shift-action $\mathbf{t}$; the costs are modified to reflect the fact that some shifts have already been performed.

Given an instance I of $\mathcal{R}$-SHIFT-BRIBERY or WEIGHTED $\mathcal{R}$-SHIFT-BRIBERY, we denote by $|I|$ the representation size of I assuming that all entries of Π (and $\mathbf{w}$, for the weighted case) are given in binary. Similarly, $|\vec{\alpha}|$ denotes the number of bits in the binary encoding of a scoring vector $\vec{\alpha}$.

3   Scoring Rules

In this section, we describe a 2-approximation algorithm that works for all scoring rules.

Theorem 3.1. *There is an algorithm $\mathcal{B}$ that given a scoring rule $\vec{\alpha} = (\alpha_1, \dots, \alpha_m)$ and an instance $I = (C, V, \Pi, p)$ of $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY with $|C| = m$, outputs a successful shift-action $\mathbf{t}$ for I that satisfies $\Pi(\mathbf{t}) \leq 2\mathrm{opt}(I)$, and runs in time $\mathrm{poly}(|I|, |\vec{\alpha}|)$.*

We split the proof of Theorem 3.1 into three steps. First (Proposition 3.3) we describe a pseudopolynomial 2-approximation algorithm $\mathcal{A}$ for our problem. Then (Proposition 3.5) we use $\mathcal{A}$ to construct another algorithm $\mathcal{A}'$, which for any $\varepsilon > 0$ produces a $(2 + \varepsilon)$-approximation and runs in time polynomial in the instance size and $\frac{1}{\varepsilon}$. Finally, we convert $\mathcal{A}'$ into a 2-approximation algorithm by bootstrapping. Throughout the proof, we fix a scoring rule $\mathcal{R}_{\vec{\alpha}} = (\alpha_1, \dots, \alpha_m)$.

We first prove a preliminary lemma that will be used to demonstrate the correctness of our approach.

Lemma 3.2. *Let $\mathbf{s} = (s_1, \dots, s_n)$ be a successful shift-action for (C, V, Π, p), and let $k = Sc_{shf(C,V,\mathbf{s})}(p) - Sc_{(C,V)}(p)$. Then every shift-action $\mathbf{r} = (r_1, \dots, r_n)$ such that $Sc_{shf(C,V,\mathbf{r})}(p) = Sc_{(C,V)}(p) + 2k$ is successful for (C, V, Π, p).*

Proof. When p is shifted from position $i + 1$ to position i in some vote $\succ^j$, he obtains $\alpha_i - \alpha_{i+1}$ extra points, while the candidate c that was in position i in $\succ^j$ prior to the shift loses $\alpha_i - \alpha_{i+1}$ points; the scores of all other candidates remain unchanged. Since $\mathbf{s}$ increases p's score by k and p wins in $shf(C, V, \mathbf{s})$, we have $\max_{c \in C}[Sc_{(C,V)}(c) - Sc_{(C,V)}(p)] \leq 2k$. Now, $\mathbf{r}$ increases p's score by $2k$ points, and does not increase the score of any other candidate, so the lemma follows. $\qquad\square$

We are now ready to implement the first step of our plan. The algorithm $\mathcal{A}$ presented in the next proposition is inspired by the 2-approximation algorithm for the Borda rule that appears in [4]; however, its analysis is substantially different.

Proposition 3.3. *There exist an algorithm $\mathcal{A}$ that given a scoring rule $\vec{\alpha} = (\alpha_1, \dots, \alpha_m)$ and an instance $I = (C, V, \Pi, p)$ of $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY with $|C| = m$, outputs a successful shift-action $\mathbf{t}$ for I that runs in time $\mathrm{poly}(|I|, |\vec{\alpha}|, \sum_{i=1}^{n} \pi^i(m))$ and satisfies $\Pi(\mathbf{t}) \leq 2\mathrm{opt}(I)$.*

Proof. Consider an instance $I = (C, V, \Pi, p)$ of $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY such that $C = \{p, c_1, \dots, c_{m-1}\}$ and $V = (v^1, \dots, v^n)$, and set $E = (C, V)$. For each integer $\ell \geq 0$, and each instance $J = (C, V', \Pi', p)$ of $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY, set $buy(J, \ell) = shf(C, V', \mathbf{t}^\ell)$, where $\mathbf{t}^\ell$ is chosen so as to maximize p's score subject to the constraint $\Pi'(\mathbf{t}^\ell) \leq \ell$, i.e., $\mathbf{t}^\ell \in \arg\max\{Sc_{shf(C,V',\mathbf{t})}(p) \mid \Pi'(\mathbf{t}) \leq \ell\}$.

procedure $\mathcal{A}(C, V, \Pi, p)$
begin
    Set $m = |C|$, $n = |V|$, $M = \sum_{i=1}^{n} \pi^i(m)$, $b = \infty$;
    for $\ell_1 = 0$ **to** M **do**
        for $\ell_2 = 0$ **to** M **do**
        begin
            $I' = buy(I, \ell_1)$;
            $I'' = buy(I', \ell_2)$;
            if p is an $\mathcal{R}_{\vec{\alpha}}$-winner in I'' **and** $\ell_1 + \ell_2 < b$ **then**
                set $b = \ell_1 + \ell_2$;
        end
    return b;
end

Figure 1: Algorithm $\mathcal{A}$.

Lemma 3.4. *For any $\ell \geq 0$, the instance $buy(I, \ell)$ is computable in time* $\mathrm{poly}(|I|, \ell)$.

Proof. For $i = 1, \ldots, n$, $k = 0, \ldots, m$, let $gain^i(k)$ denote the number of points that p gains from being shifted upwards by k positions in $\succ^i$ (or, to the top position in $\succ^i$, if k is too large).

Our algorithm relies on dynamic programming. For each $i = 0, \ldots, n$, $j = 0, \ldots, \ell$, let $F(i, j)$ denote the maximum increase in p's score that can be achieved by spending exactly j dollars on bribing the first i voters. If it is impossible to spend exactly j dollars on bribing the first i voters, set $F(i, j) = -\infty$. Naturally, we have $F(0,0) = 0$ and $F(0, j) = -\infty$ for $j = 1, \ldots, \ell$. Further, it is easy to see that for each $i = 1, \ldots, n$ and $j = 0, \ldots, \ell$ we have

$$F(i, j) = \max\{F(i - 1, j - \pi^i(k)) + gain^i(k) \mid k = 0, \ldots, m, j \geq \pi^i(k)\}.$$

Thus, we can compute $F(n, j)$ for all $j = 0, \ldots, \ell$ in time $\mathrm{poly}(|I|, \ell)$. Now pick a value $j \in \{0, \ldots, \ell\}$ that maximizes $F(n, j)$. Using standard techniques, we can find a shift-action $\mathbf{t}$ that corresponds to $F(n, j)$. Clearly, we have $buy(I, \ell) = shf(C, V, \mathbf{t})$.   (Lemma 3.4) $\square$

The pseudocode for algorithm $\mathcal{A}$ is given in Figure 1. Lemma 3.4 implies that the running time of $\mathcal{A}$ is polynomial in $|I|$ and $\sum_{i=1}^{n} \pi^i(m)$. It remains to show that $\mathcal{A}$ indeed produces a 2-approximate solution.

To this end, we will show that there exist $\ell_1, \ell_2 \leq \sum_{i=1}^{n} \pi^i(m)$ such that $I' = buy(I, \ell_1)$, $I'' = buy(I', \ell_2)$, p is an $\mathcal{R}_{\vec{\alpha}}$-winner in I'', and $\ell_1 + \ell_2 \leq 2\mathrm{opt}(I)$.

Let $\mathbf{t} = (t_1, \ldots, t_n)$ be an optimal shift-action that ensures p's victory, that is, $\Pi(\mathbf{t}) = \mathrm{opt}(I)$. Set $k = Sc_{shf(C,V,\mathbf{t})}(p) - Sc_E(p)$.

Consider an instance I' obtained from I by spending the total cost of the optimal shift action greedily, i.e., so as to maximize p's score. Formally, let $\ell_1 = \Pi(\mathbf{t})$ and set $I' =$

6

$buy(I, \ell_1)$. Let $\mathbf{s}' = (s_1', \ldots, s_n')$ be the shift-action that transforms I into $I' = (C, V', \Pi', p)$, and set $E' = (C, V')$. By construction, we have $Sc_{E'}(p) \geq Sc_E(p) + k$.

Let $\mathbf{r} = (r_1, \ldots, r_n)$ be the common part of shift-actions $\mathbf{t}$ and $\mathbf{s}'$, i.e., set $r_i = \min\{t_i, s_i'\}$ for $i = 1, \ldots, n$. Let $I^r = shf(C, V, \mathbf{r})$, where $I^r = (C, V^r, \Pi^r, p)$, and set $E^r = (C, V^r)$.

Finally, set $\ell_2 = \Pi'(\mathbf{t} - \mathbf{r})$, $I'' = buy(I', \ell_2)$, and let $\mathbf{s}'' = (s_1'', \ldots, s_n'')$ be the shift-action that transforms I' into $I'' = (C, V'', \Pi'', p)$. Let $E'' = (C, V'')$. Observe that for each $i = 1, \ldots, n$ we have either $t_i - r_i = 0$, in which case $\pi'^i(t_i - r_i) = 0$, or $t_i - r_i = t_i - s_i'$, in which case $\pi'^i(t_i - r_i) = \pi'^i(t_i - s_i') = \pi^i(t_i) - \pi^i(s_i')$. Therefore, we have $\Pi'(\mathbf{t} - \mathbf{r}) \leq \Pi(\mathbf{t})$. Now, the total cost of $\mathbf{s}' + \mathbf{s}''$ is given by $\ell_1 + \ell_2 = \Pi(\mathbf{t}) + \Pi'(\mathbf{t} - \mathbf{r}) \leq 2\Pi(\mathbf{t})$. As $\Pi(\mathbf{t}) = opt(I)$, we obtain $\ell_1 + \ell_2 \leq 2opt(I)$. It remains to show that p is a winner in $shf(C, V, \mathbf{s}' + \mathbf{s}'')$.

Set $k^r = Sc_{E^r}(p) - Sc_E(p)$. The shift-actions $\mathbf{t} - \mathbf{r}$ and $\mathbf{s}' - \mathbf{r}$ satisfy

$$Sc_{shf(C, V^r, \mathbf{t}-\mathbf{r})}(p) = Sc_{E^r}(p) + (k - k^r), \tag{1}$$
$$Sc_{shf(C, V^r, \mathbf{s}'-\mathbf{r})}(p) \geq Sc_{E^r}(p) + (k - k^r). \tag{2}$$

We have $shf(C, V^r, \mathbf{t} - \mathbf{r}) = shf(C, V, \mathbf{t})$, so p is an $\mathcal{R}_{\vec{\alpha}}$-winner in $shf(C, V^r, \mathbf{t} - \mathbf{r})$. Thus, by Lemma 3.2, any shift-action that increases the score of p in E^r by $2(k - k^r)$ points ensures that p is a winner in the resulting election. We will now show that this holds for the shift-action $\mathbf{s}'' + (\mathbf{s}' - \mathbf{r})$, and hence p is a winner in $shf(C, V^r, \mathbf{s}'' + \mathbf{s}' - \mathbf{r}) = shf(C, V, \mathbf{s}'' + \mathbf{s}')$.

For each $i = 1, \ldots, n$, if $t_i - r_i \neq 0$, then $r_i = s_i'$ and the i'th voter ranks p in the same position both in V' and in V^r. Thus, $\Pi^r(\mathbf{t} - \mathbf{r}) = \Pi'(\mathbf{t} - \mathbf{r}) = \ell_2$, and applying $\mathbf{t} - \mathbf{r}$ to I' increases p's score by the same amount as applying $\mathbf{t} - \mathbf{r}$ to I^r. By equation (1), this implies

$$Sc_{shf(C, V', \mathbf{t}-\mathbf{r})}(p) = Sc_{E'}(p) + (k - k^r). \tag{3}$$

By definition, $\mathbf{s}''$ is a shift-action of cost at most $\ell_2 = \Pi'(\mathbf{t} - \mathbf{r})$ that applied to E' increases p's score as much as possible. Thus, equation (3) implies

$$Sc_{E''}(p) \geq Sc_{E'}(p) + (k - k^r). \tag{4}$$

Since $E'' = shf(E', \mathbf{s}'')$ and $E' = shf(E^r, \mathbf{s}' - \mathbf{r})$, by combining equations (2) and (4) we obtain the following inequality:

$$\begin{aligned} Sc_{E''}(p) &\geq Sc_{E'}(p) + (k - k^r) \\ &= Sc_{shf(C, V^r, \mathbf{s}'-\mathbf{r})}(p) + (k - k^r) \\ &\geq Sc_{E^r}(p) + 2(k - k^r). \end{aligned}$$

Thus, p is a winner in election E''. This completes the proof of Proposition 3.3. $\qquad\square$

We will now convert algorithm $\mathcal{A}$ into a $(2 + \varepsilon)$-approximation scheme. A natural approach is to scale the bribery price functions given by Π by a sufficiently large parameter K. However, the appropriate choice of K depends on the actual value of the optimal solution. Therefore, we make polynomially many guesses of the value of K.

Proposition 3.5. *There exist an algorithm $\mathcal{A}'$ that given a rational $\varepsilon > 0$, a scoring rule $\vec{\alpha} = (\alpha_1, \dots, \alpha_m)$ and an instance $I = (C, V, \Pi, p)$ of $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY with $|C| = m$, runs in time $\mathrm{poly}(|I|, |\vec{\alpha}|, \frac{1}{\varepsilon})$ and outputs a successful shift-action $\mathbf{t}$ for I that satisfies $\Pi(\mathbf{t}) \leq (2 + \varepsilon)\mathrm{opt}(I)$.*

Proof. Let $R = \max_{i \in \{1, \dots, n\}} \pi^i(m)$. Algorithm $\mathcal{A}'$ executes $\lceil \log R \rceil$ iterations, each with a different value ρ, the current estimate of the cost of the most expensive shift. We start with $\rho = 1$ and we double it after every iteration. In each iteration we compute some solution—using algorithm $\mathcal{A}$—and then either discard it or keep it. After the last iteration, we pick a solution with the lowest cost. Let us now describe a single iteration.

Let ρ be the current guess of the cost of the most expensive shift. Let $K = \frac{\rho\varepsilon}{n}$. We construct an instance $J = (C, V, \Lambda, p)$, where $\Lambda = (\lambda_1, \dots, \lambda_n)$ is a sequence of shift-bribery cost functions defined as follows. For each $i = 1, \dots, n$, $j = 0, \dots, m$, we set

$$
\lambda^i(j) = \begin{cases} \left\lceil \frac{\pi^i(j)}{K} \right\rceil & \text{if } \pi^i(j) \leq \rho, \\ 2(\frac{n^2}{\varepsilon} + n) + 1 & \text{otherwise.} \end{cases}
$$

Note that all the values of functions in $(\lambda_1, \dots, \lambda_n)$ are polynomially bounded in n and $\frac{1}{\varepsilon}$. Algorithm $\mathcal{A}$ solves J in time $\mathrm{poly}(n, \frac{1}{\varepsilon})$, producing a shift-action $\mathbf{s} = (s_1, \dots, s_n)$. Note that since I and J differ only in cost functions, $\mathbf{s}$ is a valid solution for I. If we have $\lambda^i(t_i) \leq 2(\frac{n^2}{\varepsilon} + n)$ (equivalently, $\pi^i(j) \leq \rho$) for all $i = 1, \dots, n$, we store solution $\mathbf{s}$ for further use. Otherwise we discard it. Observe that in the last iteration we have $\rho \geq R$, so the solution obtained in that iteration is not discarded. After all iterations have been computed, we output the cheapest stored solution. Clearly, algorithm $\mathcal{A}'$ runs in polynomial time. It remains to show that it produces an accurate approximation.

Let $\mathbf{t} = (t_1, \dots, t_n)$ be a shift-action that corresponds to an optimal solution, and let σ be the cost of the most expensive shift within $\mathbf{t}$, i.e., set $\sigma = \max_{i=1,\dots,n} \pi^i(t_i)$. Consider an iteration in which $\frac{\rho}{2} \leq \sigma \leq \rho$. Let $J = (C, V, \Lambda, p)$ be the scaled-down instance used in this iteration. Note that $\Lambda(\mathbf{t}) \leq n \lceil \frac{\sigma}{K} \rceil \leq n \lceil \frac{\rho}{K} \rceil = n \lceil \frac{n}{\varepsilon} \rceil \leq \frac{n^2}{\varepsilon} + n$. Thus, algorithm $\mathcal{A}$ finds a solution for J with cost (in terms of Λ) at most $2(\frac{n^2}{\varepsilon} + n)$, and so this solution certainly is not discarded.

Let $\mathbf{s} = (s_1, \dots, s_n)$ be the solution produced for J by $\mathcal{A}$. As argued before, $\mathbf{s}$ is also a solution for I. We have $\Pi(\mathbf{t}) \leq \Pi(\mathbf{s}) \leq K\Lambda(\mathbf{s}) \leq 2K\Lambda(\mathbf{t})$. Indeed, the first inequality holds because $\mathbf{t}$ is an optimal solution for I, the second holds due to definition of Λ, and the third one holds because $\mathbf{s}$ is a 2-approximate solution for J. Due to rounding, for each $i = 1, \dots, n$ we have

$$
\pi^i(t_i) \leq K\lambda^i(t_i) = K \left\lceil \frac{\pi^i(t_i)}{K} \right\rceil \leq \pi^i(t_i) + K.
$$

Thus, we obtain $K\Lambda(t) \leq \Pi(\mathbf{t}) + Kn$. Using the fact that $nK = \rho\varepsilon$, we get $\Pi(\mathbf{s}) \leq 2K\Lambda(\mathbf{t}) \leq 2(\Pi(\mathbf{t}) + Kn) \leq 2\Pi(\mathbf{t}) + 2\rho\epsilon \leq (2 + 4\varepsilon)\Pi(\mathbf{t})$. The last inequality follows from the fact that $\frac{\rho}{2} \leq \Pi(\mathbf{t})$. This completes the proof of Propositon 3.5. $\qquad\square$

To complete the proof of Theorem 3.1, we will now convert our $(2 + \varepsilon)$-approximation scheme into a 2-approximation algorithm using a bootstrapping argument.

Proof of Theorem 3.1. Let $I = (C, V, \Pi, p)$ be an instance of $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY, and let $\mathbf{t} = (t_1, \ldots, t_n)$ be an optimal shift-action for I. By the pigeonhole principle, for some $i \in \{1, \ldots, n\}$ we have $\pi^i(t_i) \geq \frac{1}{n}\Pi(\mathbf{t})$. Assume for now that we know i and t_i (subsequently, we will show how to get rid of this assumption).

Let $\mathbf{d} = (0^{i-1}, t_i, 0^{m-i})$, and set $I' = shf(I, \mathbf{d})$. Clearly, we have $\mathrm{opt}(I') = \mathrm{opt}(I) - \pi^i(t_i)$. Let $\varepsilon = \frac{1}{n}$, and let $\mathbf{s} = (s_1, \ldots, s_n)$ be the shift action produced by the algorithm $\mathcal{A}'$ on (I', ε). Clearly, p is a winner in $shf(I, \mathbf{s} + \mathbf{d})$. Further, by Proposition 3.5, we have $\Pi(\mathbf{s}) \leq (2 + \varepsilon)(\Pi(\mathbf{t}) - \pi^i(t_i))$. Therefore, the cost of the shift-action $\mathbf{s} + \mathbf{d}$ can be estimated as follows:

$$
\begin{aligned}
\Pi(\mathbf{s} + \mathbf{d}) &= \Pi(\mathbf{s}) + \pi^i(t_i) \\
&\leq (2 + \varepsilon)(\Pi(\mathbf{t}) - \pi^i(t_i)) + \pi^i(t_i) \\
&\leq 2\Pi(\mathbf{t}) - \pi^i(t_i) + \varepsilon\Pi(\mathbf{t}) \\
&\leq 2\Pi(\mathbf{t}) + (\varepsilon\Pi(\mathbf{t}) - \frac{1}{n}\Pi(\mathbf{t})) = 2\Pi(\mathbf{t}),
\end{aligned}
$$

where we use the fact that $\pi^i(t_i) \geq \frac{1}{n}\Pi(\mathbf{t})$. Thus, $\mathbf{s} + \mathbf{d}$ is a 2-approximatite solution.

While we do not know the values of i and t_i, there are only n possibilities for the former and m possibilities for the latter. Thus, our algorithm $\mathcal{B}$ will try all of them, and return the best solution. Since $\frac{1}{\varepsilon} = n$, the running time of $\mathcal{B}$ is polynomial in $|I|$. $\square$

By using the algorithm $\mathcal{B}$ with $\vec{\alpha} = (m-1, \ldots, 1, 0)$, we obtain a 2-approximation algorithm for the Borda rule. This algorithm is different from the one given in [4], even though they have the same approximation guarantee. Indeed, the algorithm of [4] relies on a different dynamic programming subroutine, whose running time is polynomial in the instance size and $\sum_{i=1}^{m} \alpha_i$ (rather than the instance size and $\sum_{i=1}^{n} \pi^i(m)$, as in our construction). Since for Borda the expression $\sum_{i=1}^{m} \alpha_i$ is polynomial in the size of the instance, this immediately produces a polynomial-time algorithm. It is not hard to see that the algorithm proposed in [4] can be adapted to work for any scoring rule with $\sum_{i=1}^{m} \alpha_i = \mathrm{poly}(m)$. Of course, such an algorithm would be considerably faster than the three-step procedure of Theorem 3.1. Indeed, one may wonder if the more complicated algorithm described above is useful at all, since the scoring vectors used in practice often have small coordinates. However, an important feature of our algorithm is that it works even if each voter v^i uses his own scoring vector $\vec{\alpha}^i$. This means that we can adapt it for *weighted* voters, by replacing a voter of weight w with a scoring vector $(\alpha_1, \ldots, \alpha_m)$ by a unit-weight voter with a scoring vector $(w\alpha_1, \ldots, w\alpha_m)$.

Corollary 3.6. *There is an algorithm $\mathcal{B}^w$ that given a scoring rule $\vec{\alpha}$ and an instance $I = (C, V, \Pi, p, \mathbf{w})$ of* WEIGHTED $\mathcal{R}_{\vec{\alpha}}$-SHIFT-BRIBERY, *outputs a successful shift-action $\mathbf{t}$ for I that satisfies $\Pi(\mathbf{t}) \leq 2\mathrm{opt}(I)$, and runs in time $\mathrm{poly}(|I|, |\vec{\alpha}|)$.*

Note that there does not seem to be an easy way to derive Corollary 3.6 from the result of [4]. Indeed, Corollary 3.6 is quite surprising, as many problems in computational social choice are known to be hard for weighted voters—in fact, WEIGHTED SHIFT-BRIBERY can be shown to be NP-complete for any nontrivial family of scoring rules. On the other hand, an efficient algorithm for the weighted case is very useful, as large voter weights are ubiquitous in campaign management scenarios, where a "voter" corresponds to a collection of individuals that can be "bribed" by the same promotional activity, or in our sponsored search example, where the search terms may differ in popularity.

One may also wonder if the algorithm $\mathcal{A}$ can be simplified by using a single **for**-loop, which for each value of ℓ finds the best shift-action of cost ℓ. In Appendix A, we explore this question in more detail, showing that the resulting algorithm $\mathcal{G}$ is never better than $\mathcal{A}$, and can sometimes produce a shift-action that is almost twice as expensive as the one produced by $\mathcal{A}$. However, we do not know if $\mathcal{G}$ is nevertheless a 2-approximation algorithm for our problem.

Finally, we remark that unless P = NP, there is no FPTAS for shift bribery under scoring rules (a direct consequence of the Borda-SHIFT-BRIBERY NP-hardness proof from [4]). However, we cannot rule out the possibility that there exist approximation algorithms for Borda-SHIFT-BRIBERY whose approximation ratio is less than two.

4 Copeland and Maximin

Paper [4] shows that the decision version of the SHIFT-BRIBERY problem is NP-hard for Copeland$^\alpha$ for any rational $\alpha \in [0, 1]$ as well as for maximin. We will now give m-approximation algorithms for SHIFT-BRIBERY under Copeland$^\alpha$ with rational $\alpha \in [0, 1]$ and maximin. We then show how to improve the approximation ratio for maximin to $O(\log m)$. Finally, we argue that the $O(\log m)$ approximation ratio for maximin is asymptotically tight, and show how our results can be adapted to the weighted setting.

Under Copeland and maximin, the winner is selected on the basis of the outcomes of pairwise comparisons between the candidates. Thus, these rules can be defined in the so-called *irrational voter model*. In this model, the preferences of each voter i are given by her *preference table*, which is an antisymmetric $m \times m$ matrix whose entry (j, k) is 1 if i prefers c_j to c_k and -1 otherwise. Clearly, a preference order can be converted into a preference table, but the converse is not true, as preference tables can encode cyclic preferences.

Faliszewski et al. [7] adapt the notion of bribery to the irrational voter model; the resulting notion is known as *microbribery* (see next paragraph for a formal definition). Moreover, they show that Copeland$^\alpha$-MICROBRIBERY is in P for $\alpha \in \{0, 1\}$. We will first give a poly-time algorithm for a special case of Copeland$^\alpha$-MICROBRIBERY for all $\alpha \in [0, 1] \cap \mathbb{Q}$, and—based on this result—derive an m-approximation algorithm for Copeland$^\alpha$-SHIFT-BRIBERY. Then, we will argue that our technique can be used to convert exact algorithms for MICROBRIBERY into m-approximation algorithms for SHIFT BRIBERY for many other voting rules, including maximin. Finally, we will show an $O(\log m)$-approximation algorithm for SHIFT-BRIBERY in maximin and argue it is asymptotically optimal.

First of all, we need to describe microbribery [7] precisely. Let $\mathcal{R}$ be a voting rule defined in the irrational voter model, i.e., a mapping that for any collection of preference tables over a given set of candidates outputs a subset of the candidates. In the $\mathcal{R}$-MICROBRIBERY problem we are given an election $E = (C, V)$, where $C = \{p, c_1, \ldots, c_{m-1}\}$ and $V = (v^1, \ldots, v^n)$ is a collection of voters specified by their preference tables. Also, for each $i = 1, \ldots, n$, we have a price function $\delta^i \colon C^2 \to \mathbb{Z}^+ \cup \{+\infty\}$, where $\delta^i(c_j, c_k)$ is the cost of flipping the (j, k)-th entry in the preference table of v^i (i.e., if prior to the bribery we have $c_j \succ^i c_k$, then after the flip we obtain $c_k \succ^i c_j$, and vice versa). We require the price functions to be symmetric, i.e., for each $i = 1, \ldots, n$ and all $c_j, c_k \in C$ we require $\delta^i(c_j, c_k) = \delta^i(c_k, c_j)$. Further, we require $\delta^i(c_j, c_j) = 0$ for all $i = 1, \ldots, n$ and all $c_j \in C$. The goal is to compute a set of flips $\mathcal{S} = (S_1, \ldots, S_n)$ in the voters' preference tables that makes p an $\mathcal{R}$-winner of the election at minimum cost. We identify each flip with the pair of candidates being flipped, so we have $S_i \subseteq C \times C$ for all $i = 1, \ldots, n$. We denote an instance M of $\mathcal{R}$-MICROBRIBERY by (C, V, Δ, p), where Δ is the sequence $(\delta^1, \ldots, \delta^n)$ of microbribery price functions. Also, we let $\Delta(\mathcal{S})$ denote the total cost of flips in $\mathcal{S}$, i.e., we set

$$\Delta(\mathcal{S}) = \sum_{i=1}^{n} \sum_{(c_j, c_k) \in S_i} \delta^i(c_j, c_k).$$

Faliszewski et al. [7] have shown that Copeland$^\alpha$-MICROBRIBERY is in P for $\alpha \in \{0, 1\}$. However, it is not clear if their algorithm can be extended for all $\alpha \in [0, 1] \cap \mathbb{Q}$. Therefore, instead of using the result of [7] directly, we will now prove that Copeland$^\alpha$-MICROBRIBERY is in P for all $\alpha \in [0, 1] \cap \mathbb{Q}$ as long as the only permissible flips are the ones that involve the manipulator's preferred candidate p. We will then show that for all $\alpha \in [0, 1] \cap \mathbb{Q}$, finding an m-approximate solution for Copeland$^\alpha$-SHIFT-BRIBERY can be reduced to solving an instance of Copeland$^\alpha$-MICROBRIBERY that satisfies this constraint.

Lemma 4.1. *For any $\alpha \in [0, 1] \cap \mathbb{Q}$, Copeland$^\alpha$-MICROBRIBERY is in P for all instances $M = (C, V, \Delta, p)$ with $|C| = m$, $|V| = n$ such that for any $i = 1, \ldots, n$ and any two $c_j, c_k \in C$, it holds that if $p \notin \{c_j, c_k\}$ then $\delta^i(c_j, c_k) = +\infty$.*

Proof. Fix $\alpha \in [0, 1] \cap \mathbb{Q}$, and let $M = (C, V, \Delta, p)$ be our input instance of Copeland$^\alpha$-MICROBRIBERY. Assume that $C = \{p, c_1, \ldots, c_{m-1}\}$, and let $E = (C, V)$.

Our algorithm works as follows. For all $i = 0, \ldots, m$ and all $j = 0, \ldots, m$, we compute $k(i, j) = i + \alpha j$, and check whether $Sc_E(p) \leq k(i, j) \leq m - 1$. If this condition is satisfied, we find a minimum-cost microbribery $\mathcal{S}(i, j)$ that ensures that p has $k(i, j)$ points and every other candidate has at most $k(i, j)$ points; if no such microbribery exists, we declare the cost of $\mathcal{S}(i, j)$ to be $+\infty$. In the end, we output the cheapest microbribery found. Observe that setting $i = m - 1$, $j = 0$ results in a successful microbribery of finite cost in which p wins every pairwise election. Thus, we are guaranteed to output a microbribery of finite cost. This approach is clearly correct and it remains to show how to compute the microbribery $\mathcal{S}(i, j)$ for each pair (i, j).

Let us fix a pair (i, j) such that $Sc_E(p) \leq k(i, j) \leq m - 1$. Since we can only flip entries of preference tables that involve p, for each candidate $c_k \in C$ we can decrease c_k's score by

11

either 1 or $1 - \alpha$ (if c_k wins the pairwise election with p) or by α (if c_k ties the pairwise election with p). In each case it is easy to compute the cheapest way of achieving this. Thus, for each $c_k \in C$, if $Sc_E(c_k) > k(i,j)$, we perform the cheapest microbribery that involves preference table entries for p and c_k and brings c_k's score down to $k(i,j)$ or below; clearly, if this is impossible, then for this pair (i,j) a microbribery with the required properties does not exist. After this step, each candidate $c_k \in C \setminus \{p\}$ has at most $k(i,j)$ points and it remains to find the cheapest microbribery that ensures that p has exactly $k(i,j)$ points. This can easily be done by using standard dynamic programming techniques. $\qquad\square$

We will now use Lemma 4.1 to obtain an m-approximation algorithm for SHIFT-BRIBERY under Copeland$^\alpha$, where $\alpha \in [0,1] \cap \mathbb{Q}$.

Theorem 4.2. *There exists a poly-time algorithm that given an instance $I = (C, V, \Pi, p)$ of* Copeland$^\alpha$-SHIFT-BRIBERY *with $\alpha \in [0,1] \cap \mathbb{Q}$ and $|C| = m$ outputs a shift-action $\mathbf{s}$ such that p is a winner in $shf(C, V, \mathbf{s})$ and $\Pi(\mathbf{s}) \le m \cdot opt(I)$.*

Proof. Fix an $\alpha \in [0,1] \cap \mathbb{Q}$ and an instance $I = (C, V, \Pi, p)$ of Copeland$^\alpha$-SHIFT-BRIBERY with $C = \{p, c_1, \ldots, c_{m-1}\}$, $V = (v^1, \ldots, v^n)$. For each $i = 1, \ldots, n$, assume that the preference order of a voter v^i is given by

$$c_{j_{i,k(i)}} \succ c_{j_{i,k(i)-1}} \succ \cdots \succ c_{j_{i,1}} \succ p \succ \cdots .$$

Our algorithm first converts I into an instance $M = (C, \hat{V}, \Delta, p)$ of Copeland$^\alpha$-MICROBRIBERY. The list $\hat{V}$ contains n voters. The preference table of each voter $\hat{v}^i$ is constructed from the preference ordering of v^i: we set the (j,k)-th entry of $\hat{v}^i$'s preference table to 1 if and only if $c_j \succ^i c_k$.

The price functions $\Delta = (\delta^1, \ldots, \delta^n)$ are defined as follows. For each $i = 1, \ldots, n$ and all $\ell = 1, \ldots, k(i)$, we set $\delta^i(p, c_{j_{i,\ell}}) = \delta^i(c_{j_{i,\ell}}, p) = \pi^i(\ell)$. For all other pairs of candidates, we set the value of δ^i to be ∞.

Note that the resulting instance of Copeland$^\alpha$-MICROBRIBERY satisfies the conditions of Lemma 4.1. Therefore, we can use Lemma 4.1 to compute an optimal solution to M. It is easy to see that there is a solution to M which does not involve flipping any of the preference-table entries with cost ∞. Therefore, an optimal solution to M is a sequence $\mathcal{S} = (S_1, \ldots, S_n)$, where for each $i = 1, \ldots, n$ the set S_i consists of flips that involve p and candidates in $c_{j_{i,1}}, \ldots, c_{j_{i,k(i)}}$ only.

We derive a shift-action $\mathbf{s} = (s_1, \ldots, s_n)$ from $\mathcal{S}$ as follows. For each $i = 1, \ldots, n$, we set $s_i = 0$ if $S_i \cap \{c_{j_{i,1}}, \ldots, c_{j_{i,k(i)}}\} = \emptyset$, and $s_i = \max\{\ell \mid c_{j_{i,\ell}} \in S_i'\}$ otherwise. It is easy to see that since $\mathcal{S}$ is a solution for M, $\mathbf{s}$ is a shift-action that ensures that p is a winner in I. Further, we have

$$\Pi(\mathbf{s}) \le \Delta(\mathcal{S}). \tag{5}$$

Our algorithm outputs $\mathbf{s}$. It remains to show that $\Delta(\mathcal{S}) \le m \cdot opt(I)$.

Let $\mathbf{t} = (t_1, \ldots, t_n)$ be an optimal-cost shift-action that ensures p's victory in I. From $\mathbf{t}$, we can derive a solution $\mathcal{T} = (T_1, \ldots, T_n)$ to M as follows. For each $i = 1, \ldots, n$, if $t_i = 0$,

we set $T_i = \emptyset$, and otherwise we set

$$T_i = \{(p, c_{j_{i,1}}), (p, c_{j_{i,2}}), \ldots, (p, c_{j_{i,t_i}})\}.$$

Since applying the shift-action $\mathbf{t}$ to I makes p a winner, applying $\mathcal{T}$ to M makes p a winner in M as well. Naturally, since $\mathcal{S}$ is an optimal solution for M, we have

$$\Delta(\mathcal{S}) \le \Delta(\mathcal{T}). \tag{6}$$

For each $i = 1, \ldots, n$, let $\delta^i(T_i) = \sum_{(p,c) \in T_i} \delta^i(p, c)$. We have

$$\delta^i(T_i) = \sum_{(p,c) \in T_i} \delta^i(p, c) = \sum_{\ell=1}^{t_i} \delta^i(p, c_{j_{i,\ell}}) = \sum_{\ell=1}^{t_i} \pi^i(\ell) \le t_i \pi^i(t_i) \le m\pi^i(t_i). \tag{7}$$

Consequently, $\Delta(\mathcal{T}) \le m\Pi(\mathbf{t})$. Combining this inequality with (5) and (6), we obtain

$$\Pi(\mathbf{s}) \le \Delta(\mathcal{S}) \le \Delta(\mathcal{T}) \le m\Pi(\mathbf{t}).$$

Since $\mathbf{t}$ is an optimal solution for I, the shift-action $\mathbf{s}$ is an m-approximate solution for I. The algorithm clearly works in polynomial time. $\square$

The proof of Theorem 4.2 can also be adapted for maximin (using the result of [4], where the authors show that maximin-MICROBRIBERY is in P), and, more generally, for any other rule that is defined for irrational voters, has a polynomial-time microbribery algorithm (at least for the instances constructed in the proof), and satisfies a certain form of monotonicity (we will not formalize this notion of monotonicity; essentially, the voting rule has to guarantee that our algorithm's translations between microbribery solutions and shift-bribery solutions keep p a winner).

To extend these approximability results for Copeland and maximin to the case of weighted voters, we use essentially the same strategy as in the proof of Theorem 3.1. That is, we first show that both WEIGHTED Copeland-MICROBRIBERY and WEIGHTED Maximin-MICROBRIBERY admit an FPTAS as long as the input instances satisfy the condition of Lemma 4.1. We then use this FPTAS as an oracle inside the algorithm described in the proof of Theorem 4.2 to obtain an $m(1 + \varepsilon)$-approximation scheme for our problems. We then use the idea presented in the proof of Theorem 3.1 to convert this approximation scheme into an m-approximation algorithm. As in the proof of Proposition 3.5, the FP-TAS proceeds by executing $\lceil \log \sum_{i=1}^n \sum_{j=1}^{m-1} \delta^i(p, c_j) \rceil$ iterations that correspond to different guesses of the cost of the optimal microbribery. In iteration i, we set $\rho = 2^{i-1}$, and round all bribery prices that do not exceed ρ up to the nearest multiple of $\varepsilon\rho$, where ε is the given error parameter; all prices that exceed ρ are set to $+\infty$. We then find an optimal microbribery $\mathcal{S}(\rho)$ for the rounded prices, and discard it if its cost is too large relative to ρ. For the rounded prices, the optimal microbribery for both Copeland and maximin is easy to find, and the cost of the best non-discarded microbribery provides a good approximation to the cost of the optimal microbribery.

Interestingly, our FPTAS for Copeland$^\alpha$ only works for the instances of WEIGHTED Copeland$^\alpha$-MICROBRIBERY that satisfy the condition of Lemma 4.1. Indeed we can show that, in general, WEIGHTED Copeland$^\alpha$-MICROBRIBERY is inapproximable up to any factor for all rational $\alpha < 1$. This result holds even if there are only three candidates.

Proposition 4.3. *For all rational $\alpha < 1$ and any K, there does not exist a poly-time K-approximation algorithm for WEIGHTED Copeland$^\alpha$-MICROBRIBERY unless $P = NP$.*

Proof. The reduction is from PARTITION. An instance of PARTITION is given by a list of n positive integers $A = (a_1, \ldots, a_n)$. It is a "yes"-instance if there is a set of indices $J \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in J} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$, and a "no"-instance otherwise.

Given an instance $A = (a_1, \ldots, a_s)$ of PARTITION, we set $\sum_{i=1}^{n} a_i = 2B$, and construct an instance of WEIGHTED Copeland$^\alpha$-MICROBRIBERY as follows. There are $s + 2$ voters $(v^1, \ldots, v^s, v^{s+1}, v^{s+2})$ with weights $(a_1, \ldots, a_s, B, B)$, and three candidates a, b and p, where p is the briber's preferred candidate. The voter's preferences are set as follows. Voters v^{s+1} and v^{s+2} prefer p to any other candidate; further, v^{s+1} prefers a to b, and v^{s+2} prefers b to a. All other voters prefer a and b to p, and a to b. Further, for voters v^{s+1} and v^{s+2} the cost of any flip is $+\infty$, while for $i = 1, \ldots, s$ we have $\delta^i(a, b) = 0$, $\delta^i(a, p) = \delta^i(b, p) = +\infty$.

In this instance, p's score is 2α, a's score is $\alpha + 1$, and b's score is α. The only way for p to win is to ensure that all candidates have 2α points, i.e., there is a tie between a and b. Clearly, this can only be done if we started with a "yes"-instance of PARTITION. □

For the remainder of this section let us return to unweighted shift bribery and maximin. We have already shown that maximin-SHIFT BRIBERY can be solved in polynomial-time with a linear approximation ratio, and now we will show that, in fact, a logarithmic one is possible and asymptotically optimal. The most important building block of our new maximin algorithm is a result of Caragiannis et al. [3] proved in the context of computing the Dodgson score. Thus, let us now describe this result and then show how it can be applied to maximin-SHIFT-BRIBERY.

Given an election $E = (C, V)$, the *Dodgson score* of a candidate $c \in C$ is the minimum number of positions by which c needs to be shifted upwards in the preference orders of the voters in V to become a Condorcet winner. Observe that the Dodgson score of a candidate is exactly the cost of shift bribery that makes c a Condorcet winner, assuming that each unit shift has a unit cost.

It is known that determining whether a candidate is a winner in Dodgson elections is an NP-hard problem [1] (in fact, it is Θ_2^p-complete [8]). Caragiannis et al. [3] gave a polynomial-time $O(\log m)$-approximation algorithm for computing Dodgson scores. In fact, their algorithm is somewhat more general and the next theorem is a direct consequence of the results in [3].

Theorem 4.4. *There is an algorithm $\mathcal{F}$ that given an instance $I = (C, V, \Pi, p)$ of $\mathcal{R}$-SHIFT-BRIBERY with $|C| = m$ and a sequence $k_1, \ldots, k_{m-1}$ of nonnegative integers, computes a shift-action $\mathbf{t}$ that has the following properties:*

14

1. *For each $i = 1, \ldots, m - 1$, $N_{shf(C,V,\mathbf{t})}(p, c_i) \geq \min(N_{(C,V)}(p, c_i) + k_i, |V|)$.*

2. *$\Pi(\mathbf{t}) = O(\log m)\Pi(\mathbf{s})$, where $\mathbf{s}$ is a minimal-cost shift-action that satisfies the above condition.*

We can now use Theorem 4.4 to give a polynomial-time $O(\log m)$-approximation algorithm for maximin-SHIFT-BRIBERY. The main idea of our approach is that, for a given instance I of maximin-SHIFT-BRIBERY, it is easy to provide a description of an optimal shift-action in the format of Theorem 4.4.

Theorem 4.5. *There exists a poly-time algorithm that given an instance $I = (C, V, \Pi, p)$ of maximin-SHIFT-BRIBERY with $|C| = m$ outputs a shift-action $\mathbf{s}$ such that p is a winner in $shf(C, V, \mathbf{s})$ and $\Pi(\mathbf{s}) = O(\log m) \cdot \mathrm{opt}(I)$.*

Proof. Let I be our input instance as in the statement of the theorem and let $E = (C, V)$. Our algorithm executes a series of iterations, one for each value k, $Sc_E(p) \leq k \leq |V|$. The goal of the iteration for value k is to find a shift-action $\mathbf{r}^k = (r_1^k, \ldots, r_n^k)$ that (a) ensures that the score of p is at least k and the score of each candidate $c \in C \setminus \{p\}$ is at most k, and (b) $\Pi(\mathbf{r}^k)$ is within $O(\log m)$ of the cost of an optimal-cost shift-action that achieves (a). Given the shift-actions produced in these iterations, the algorithm outputs one with the lowest cost. Since the optimal shift-action for I is an optimal shift-action for one of the iterations, this algorithm outputs an $O(\log m)$-approximate solution for I. It remains to show how to execute the iterations.

Let us now fix value k, $Sc_E(p) \leq k \leq |V|$, and describe the iteration for k. Let us assume that $C = \{p, c_1, \ldots, c_{m-1}\}$. Since a shift-action can move p only, via applying some shift action $\mathbf{r} = (r_1, \ldots, r_n)$ to E we can affect each of the values $N_E(p, c_1), \ldots, N_E(p, c_{m-1})$ but neither of the values $N_E(c_i, c_j)$ for $c_i, c_j \in C$.

Thus, shift-action $\mathbf{r}^k$ has to satisfy the following conditions. First, for each $i = 1, \ldots, m-1$ we require $N_{shf(C,V,\mathbf{r}^k)}(p, c_i) \geq k$. This guarantees that the maximin score of p is at least k. Second, for each $c_i \in C \setminus \{p\}$ such that $Sc_E(c_i) > k$, we require $N_{shf(C,V,\mathbf{r}^k)}(c_i, p) < k$, which is equivalent to demanding that $N_{shf(C,V,\mathbf{r}^k)}(p, c_i) \geq |V| - k$. This guarantees that the score of each candidate in $C \setminus \{p\}$ is at most k. We see that these two requirements can easily be phrased in terms of the input to the algorithm of Theorem 4.4. Thus, we compute $\mathbf{r}^k$ using Theorem 4.4. Clearly, the resulting shift action $\mathbf{r}^k$ satisfies our requirements. Since $\mathbf{r}^k$ can be computed in polynomial time, the proof is complete. □

We remark that the approximation guaranteee given by Theorem 4.5 is asymptotically optimal. This follows from the fact that the reduction of EXACT-COVER-BY-3-SETS to maximin-SHIFT-BRIBERY given in [4] can be modified to reduce from SET-COVER, in a way that allows maximin-SHIFT-BRIBERY to inherit the inapproximability properties of SET-COVER (see [9] for inapproximability results for SET-COVER).

5 Inapproximability of Swap Bribery

In contrast to shift bribery, swap bribery is hard to approximate up to an arbitrary factor for all voting rules for which the possible winner problem is hard; this includes almost all scoring rules, and, in particular, k-approval for $k \geq 2$ and Borda [2], Copeland [10], and maximin [10]. Indeed, in the reduction from the possible winner problem to swap bribery, the resulting instance of swap bribery has a bribery of cost 0 if and only if the original instance of the possible winner problem is a "yes"-instance. Thus, *any* approximation algorithm for swap bribery can be used to decide the possible winner problem.

6 Conclusions

We have presented approximation algorithms for campaign management under a number of voting rules. Most of our results hold even for weighted voters. We believe that designing algorithms for the case of weighted voters is important, since in realistic campaign management scenarios a "voter" to be bribed is usually a group of voters that can be reached by the same ad. By the same token, it would be interesting to extend our results to settings where we can reach several *non-identical* voters with the same ad; this would correspond to shift bribery with "bulk discounts". Another, more applied direction would be to identify commercial campaign management scenarios (where candidates correspond to services or products) that can be handled using our model; the sponsored search example is the introduction is the first step in that direction. Finally, a natural direction for further study is to design efficient algorithms for shift bribery with better approximation ratios, or to prove that our results are (asymptotically) optimal.

References

[1] J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[2] N. Betzler and B. Dorn. Towards a dichotomy of finding possible winners in elections based on scoring rules. In *Proc. of MFCS'09*, pp. 124–136, 2009.

[3] I. Caragiannis, J. Covey, M. Feldman, C. Homan, C. Kaklamanis, N. Karanikolas, A. Procaccia, and J. Rosenschein. On the approximability of Dodgson and Young elections. In *Proc. of SODA'09*, pp. 1058–1067, 2009.

[4] E. Elkind, P. Faliszewski, and A. Slinko. Swap bribery. In *Proc. of SAGT'09*, pp. 299–310, 2009.

[5] P. Faliszewski. Nonuniform bribery (short paper). In *Proc. of AAMAS'08*, pp. 1569–1572, 2008.

[6] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35:485–532, 2009.

[7] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence Research*, 35:275–341, 2009.

[8] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.

[9] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. In *Proc. of STOC'97*, pp. 475–484, 1997.

[10] L. Xia and V. Conitzer. Determining possible and necessary winners under common voting rules given partial orders. In *Proceedings of AAAI'08*, pp. 196–201, 2008.

[11] L. Xia, V. Conitzer, A. Procaccia, and J. Rosenschein. Complexity of unweighted manipulation under some common voting rules. In *Proc. of IJCAI'09*, pp. 348–353, 2009.

A Analysis of a single-pass variant of $\mathcal{A}$

A counterintuitive property of the algorithm $\mathcal{A}$ described in Section 3 is that it uses two **for**-loops instead of one. That is, it splits the money to be spent into two parts ℓ_1 and ℓ_2, greedily selects a shift-action $\mathbf{s}'$ that maximizes p's score given the budget ℓ_1, and then greedily selects a shift-action $\mathbf{s}''$ for $shf(C, V, \mathbf{s}')$ given the budget ℓ_2. It is natural to ask if a "greedy" single-pass algorithm given in Figure 2 is still a 2-approximation algorithm for our problem.

It is easy to see that each shift-action considered by $\mathcal{G}$ is also considered by $\mathcal{A}$, so $\mathcal{G}$ never produces a better solution than $\mathcal{A}$. However, the converse is not true. Indeed, we will now show that the solution produced by $\mathcal{G}$ can be almost twice as expensive as the one produced by $\mathcal{A}$.

Theorem A.1. *For any $\varepsilon > 0$, there is an instance $I = (C, V, \Pi, p)$ of* Borda-shift-bribery *such that $\Pi(\mathcal{G}(I)) \geq (2 - \varepsilon)\Pi(\mathcal{A}(I))$.*

```
procedure 𝒢(C, V, Π, p)
begin
    Set m = |C|, n = |V|, M = ∑ⁿᵢ₌₁ πⁱ(m), b = ∞;
    for ℓ = 0 to M do
    begin
        I' = buy(I, ℓ);
        if p is an ℛ-winner in I' and ℓ < b then
            b = ℓ;
    end
    return b;
end
```

Figure 2: Algorithm $\mathcal{G}$.

Proof. Given an $\varepsilon > 0$, set $k = \lceil \frac{1}{\varepsilon} \rceil$, and let $T = 2k$. We will now construct an instance I of Borda-SHIFT-BRIBERY with $\mathrm{opt}(I) = 2kT$, such that $\mathcal{A}(I)$ outputs an optimal solution, while $\mathcal{G}(I)$ finds a solution of cost $4kT - 3k$. As we have $4kT - 3k > 4kT - 4k = 2kT(2 - \frac{2}{T}) > 2kT(2 - \varepsilon)$, this suffices to prove the theorem.

To construct the instance $I = (C, V, \Pi, p)$, we set $C = \{p, c, a_1, \dots, a_{4k}\}$ and $V = (v^1, \dots, v^{4k+2})$. The voters have the following preferences. For $i = 1, \dots, 2k$, v^{2i-1} has preference order $c \succ p \succ a_1 \succ a_2 \succ \cdots \succ a_{4k}$ and v^{2i} has preference order $c \succ p \succ a_{4k} \succ a_{4k-1} \succ \cdots \succ a_1$. Voters v^{4k+1} and v^{4k+2} have preference orders $c \succ a_{4k} \succ a_{4k-1} \succ \cdots \succ a_1 \succ p$ and $p \succ a_1 \succ a_2 \succ \cdots \succ a_{4k} \succ c$, respectively. Let $E = (C, V)$. It is easy to verify that the candidates have the following Borda scores: $Sc_E(p) = 16k^2 + 4k + 1$, $Sc_E(c) = (16k^2 + 4k + 1) + 4k$, and $Sc_E(a_i) = 8k^2 + 2k + 1$ for $i = 1, \dots, 4k$. That is, c beats p by $4k$ points, but p has more points than any other candidate.

We will now define the sequence $\Pi = (\pi^1, \dots, \pi^{4k+2})$ of price functions. Since the voters $v^1, \dots, v^{4k}$ rank p second, we can fully specify π^i for $i = 1, \dots, 4k$ by setting $\pi^i(1) = T$. Similarly, since v^{4k+2} ranks p first, the function π^i is fully described by setting $\pi^{4k+2}(0) = 0$. The price function π^{4k+1} is defined as follows. We have $\pi^{4k+1}(0) = 0$ and for each j, $1 \le j \le 4k + 1$, it holds that:

$$
\pi^{4k+1}(j) - \pi^{4k+1}(j-1) = \begin{cases} 0 & \text{if } j = 0, \\ T+1 & \text{if } j = 1, \\ T & \text{if } 2 \le j \le k \\ T-2 & \text{if } j = k+1 \\ T-1 & \text{if } k+2 \le j \le 4k+1 \end{cases}
$$

It is easy to see that minimal-cost solutions involve shifting our preferred candidate p upwards by one position in the preference orders of $2k$ voters among those in $V' = (v^1, \dots, v^{4k})$: each such shift gives p one extra point and takes away one point from c. Thus, $\mathrm{opt}(I) = 2kT$.

18

Algorithm $\mathcal{A}$ successfully finds an optimal solution. Indeed, any shift-action of cost kT can increase p's score in I by at most k and to achieve this, we need to shift p upwards by one position in some k votes chosen from V'. Let I' be the instance obtained by this action. In I', the shift-action of cost kT that maximizes p's score is obtained by shifting p upwards by one position in the votes of the k "unshifted" voters from V'. Thus, for $\ell_1 = \ell_2 = kT$, $\mathcal{A}$ finds an optimal solution.

On the other hand, algorithm $\mathcal{G}$ outputs the shift-action $(0, \ldots, 0, 4k, 0)$ of cost $T + 1 + (k-1)T + 3k(T-1) - 1 = 4kT - 3k$. Indeed, if $\ell \leq kT$, the subroutine $buy(I, \ell)$ shifts p upwards by one position in the preference orders of $\lfloor \frac{\ell}{T} \rfloor$ voters among V'. On the other hand, if $\ell > kT$, $buy(I, \ell)$ shifts p upwards as much as possible given the budget ℓ in the preference order of v^{4k+1} (recall that $buy(I, \ell)$ selects the cheapest shift-action among those that maximize p's score without exceeding cost ℓ). Moreover, by Lemma 3.2, the shift $(0, \ldots, 0, 4k, 0)$ makes p a winner, as it increases its score by $4k$ points, whereas the optimal solution increases p's score by $2k$ points. $\qquad\square$

Note that for the family of instances given in the proof of Theorem A.1, $\mathcal{A}$ actually produces an optimal solution, so $\mathcal{G}$ still provides a factor 2 approximation in this case. In fact, it is not clear whether $\mathcal{G}$ is a 2-approximation algorithm for shift bribery with respect to the class of all scoring rules. Note, however, that $\mathcal{G}$ may behave very differently from $\mathcal{A}$: in the proof of Theorem A.1, the shift-actions produced by $\mathcal{G}$ and $\mathcal{A}$ are completely disjoint. Thus, even if $\mathcal{G}$ happens to be a 2-approximation algorithm for our problem, the proof that it achieves this approximation ratio is very likely to require additional insights.